Needham-Schroeder public-key の ProVerif による評価結果

国立研究開発法人 情報通信研究機構

1. 基本情報

- ✧ 名前

  Needham-Schroeder public-key

- ✧ 機能

  公開鍵サーバを用いた相互認証プロトコル。

- ✧ 関連する文書

  R.Needham and M.Schroeder, "Using encryption for authentication in large networks of computers," Communications of the ACM, 21(12), December 1978.

2. ProVerif の文法による記述

2.1. プロトコル仕様

```
(*
set ignoreTypes = attacker.
*)

free c: channel.

type host.
type nonce.
type pkey.
type skey.
type spkey.
type sskey.

fun nonce_to_bitstring(nonce): bitstring [data,typeConverter].

(* Public key encryption *)
```

```
fun pk(skey): pkey.
fun encrypt(bitstring, pkey): bitstring.
reduc forall x: bitstring, y: skey; decrypt(encrypt(x,pk(y)),y) = x.

(* Signatures *)
fun spk(sskey): spkey.
fun sign(bitstring, sskey): bitstring.
reduc forall m: bitstring, k: sskey; getmess(sign(m,k)) = m.
reduc forall m: bitstring, k: sskey; checksign(sign(m,k), spk(k)) = m.

(* Shared key encryption *)
fun sencrypt(bitstring,nonce): bitstring.
reduc forall x: bitstring, y: nonce; sdecrypt(sencrypt(x,y),y) = x.

(* Secrecy assumptions *)
not attacker(new skA).
not attacker(new skB).
not attacker(new sskS).

(* 3 honest host names hostA and hostB and hostS *)
free hostA, hostB, hostS: host.

table pke_keys(host, pkey, skey).
table sign_keys(host, spkey, sskey).
table honest(host).

(* Queries *)
free secretANa, secretANb, secretBNa, secretBNb: bitstring [private].
event beginA(host, host, nonce, nonce).
event endA(host, host, nonce, nonce).
event beginB(host, host, nonce, nonce).
event endB(host, host, nonce, nonce).
```

```
(* 1 *)
query x: host, y: host, nx: nonce, ny: nonce;
      inj-event(endB(x, y, nx, ny)) ==> inj-event(beginA(x, y, nx, ny)).

(* 2 *)
query x: host, y: host, nx: nonce, ny: nonce;
      inj-event(endA(x, y, nx, ny)) ==> inj-event(beginB(x, y, nx, ny)).

(* 3 *)
query attacker(secretANa);
      attacker(secretANb);
      attacker(secretBNa);
      attacker(secretBNb).

(*
free secret_test: bitstring [private].
query attacker(secret_test).
*)

let processA =
        in(c, (A: host, B: host, S: host));
        if S = hostS then
         if A <> B then
         get honest(=A) in
         (* Message 1: Get the public key certificate for the other host
*)
         out(c, (A, B));
         (* Message 2 *)
         in(c, m2: bitstring);
         get sign_keys(=S, spkS, sskS) in
        let m21 = checksign(m2, spkS) in
```

```
        let (m211: pkey, m212: host) = m21 in
        let pkB = m211 in
        if B = m212 then
        (* Message 3 *)
         new Na: nonce;
        out(c, encrypt((Na, A), pkB));
        (* Message 6 *)
        in(c, m6: bitstring);
        get pke_keys(=A, pkA, skA) in
        let m61 = decrypt(m6, skA) in
        let (m611: nonce, m612: nonce) = m61 in
        if m611 = Na then
        let Nb = m612 in
         event beginA(A, B, Na, Nb);
        (* Message 7 *)
        out(c, encrypt(nonce_to_bitstring(Nb), pkB));
        (* Security *)
         get honest(=B) in
         event endA(A, B, Na, Nb);
         out(c, sencrypt(secretANa, Na));
         out(c, sencrypt(secretANb, Nb)).

let processB =
        in(c, (A: host, B: host, S: host));
        if S = hostS then
         get honest(=A) in
        (* Message 3 *)
         in(c, m3: bitstring);
        get pke_keys(=B, pkB, skB) in
         let m31 = decrypt(m3, skB) in
        let (m311: nonce, m312: host) = m31 in
        let Na = m311 in
```

```
        if A = m312 then
         (* Message 4: Get the public key certificate for the other host
*)
        out(c, (B, A));
         (* Message 5 *)
         in(c, m5: bitstring);
        get sign_keys(=S, spkS, sskS) in
        let m51 = checksign(m5, spkS) in
        let (m511: pkey, m512: host) = m51 in
        let pkA = m511 in
        if B = m512 then
        (* Message 6 *)
         new Nb: nonce;
         event beginB(A, B, Na, Nb);
         out(c, encrypt((Na, Nb), pkA));
        (* Message 7 *)
         in(c, m7: bitstring);
        if nonce_to_bitstring(Nb) = decrypt(m7, skB) then
         (* Security *)
         get honest(=A) in
         get honest(=S) in
         event endB(A, B, Na, Nb);
         out(c, sencrypt(secretBNa, Na));
         out(c, sencrypt(secretBNb, Nb)).

(* Server *)

let processS =
        in(c,(A: host, B: host, S: host));
         get honest(=S) in
        (* Message1 *)
         in(c, m1: bitstring);
```

```
         let (m11: host, m12: host) = m1 in
         if A = m11 then
         if B = m12 then
        (* Message2 *)
         get sign_keys(=S, spkS, sskS) in
         get pke_keys(=B, pkB, skB) in
        out(c,sign((pkB,B),sskS));
        (* Message4 *)
         in(c, m4: bitstring);
         let (m41: host, m42: host) = m4 in
         if B = m41 then
         if A = m42 then
        (* Message5 *)
         get pke_keys(=A, pkA, skA) in
                (* 参加者 A の公開鍵をテーブルから参照して pkA に代入
*)
        out(c,sign((pkA,A),sskS)).

(* Key registration *)
let regist_pke_keys =
        in(c, (X: host, pkX: pkey, skX: skey));
        if X <> hostA && X <> hostB && X <> hostS
         then insert pke_keys(X, pkX, skX).

let regist_sign_keys =
        in(c, (X: host, spkX: spkey, sskX: sskey));
        if X <> hostA && X <> hostB && X <> hostS
         then insert sign_keys(X, spkX, sskX).



process new skA: skey;
         let pkA = pk(skA) in
```

```
        out(c, pkA);
        insert honest(hostA);
         insert pke_keys(hostA, pkA, skA);
         (**)
        new skB: skey;
         let pkB = pk(skB) in
        out(c, pkB);
        insert honest(hostB);
         insert pke_keys(hostB, pkB, skB);
         (**)
         new sskS: sskey;
         let spkS = spk(sskS) in
         out(c, spkS);
        insert honest(hostS);
        insert sign_keys(hostS, spkS, sskS);
         (
          (!processA) |
          (!processB) |
          (!processS) |
           (!regist_sign_keys) |
           (!regist_pke_keys)
        )
```

参加者は自分自身を相手して暗号プロトコルを実行しないと仮定した。

2.2. 攻撃者モデル

上述の記述に含まれる。

2.3. セキュリティ要件

上述の記述の (* queries *) に該当する。

```
(* (1) *)
query x: host, y: host, nx: nonce, ny: nonce;
      inj-event(endB(x, y, nx, ny)) ==> inj-event(beginA(x, y, nx, ny)).


(* (2) *)
```

```
query x: host, y: host, nx: nonce, ny: nonce;
      inj-event(endA(x, y, nx, ny)) ==> inj-event(beginB(x, y, nx, ny)).

(* (3) *)
query attacker(secretANa);
      attacker(secretANb);
      attacker(secretBNa);
      attacker(secretBNb).
```

(1) 参加者 B による参加者 A の認証。具体的には参加者 B がパラメータ A, B, Na, Nb を用いて実行を完了したならば、同じパラメータを用いて通信を行なった参加者 A が存在する。

(2) 参加者 A による参加者 B の認証。具体的には参加者 A がパラメータ A, B, Na, Nb を用いて実行を完了したならば、同じパラメータを用いて通信を行なった参加者 B が存在する。

(3) 参加者が生成したノンス Na, Nb の秘匿性。

3. ProVerif による評価結果

3.1. 出力

セキュリティ要件のうち、(2)のみ成り立ち（true)、(1)及び(3)については成り立たず（false)、攻撃が発見された。

```
RESULT not attacker(secretANa[]) is true.
RESULT not attacker(secretANb[]) is true.
RESULT not attacker(secretBNa[]) is false.
RESULT not attacker(secretBNb[]) is false.
RESULT inj-event(endA(x_21062,y_21063,nx,ny)) ==>
 inj-event(beginB(x_21062,y_21063,nx,ny)) is true.
RESULT inj-event(endB(x_59796,y_59797,nx_59798,ny_59799)) ==>
 inj-event(beginA(x_59796,y_59797,nx_59798,ny_59799)) is false.
RESULT (even event(endB(x_89211,y_89212,nx_89213,ny_89214)) ==>
 event(beginA(x_89211,y_89212,nx_89213,ny_89214)) is false.)
```

3.2. 攻撃の解説

以下の図 1.に示すとおり、Lowe が発見した攻撃が検出された。この攻撃では、参加者 A

が攻撃者と暗号プロトコルの実行を開始したことを利用して、攻撃者は参加者 B に対して参加者 A のふりをすることができる。

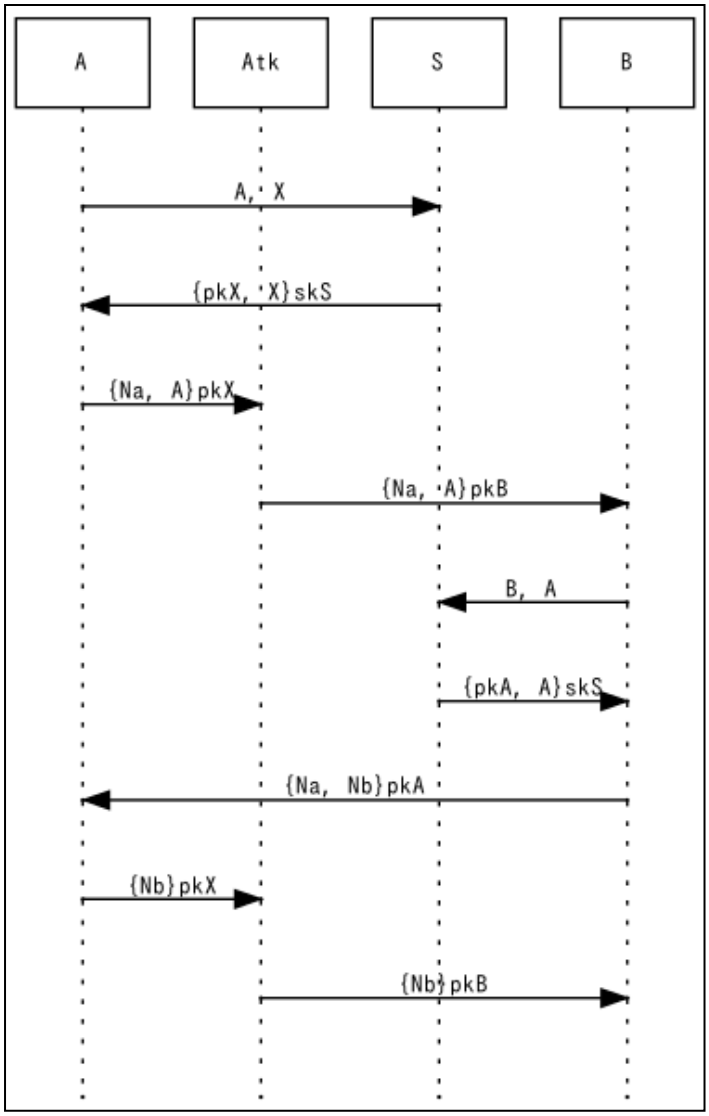

図 1. 攻撃シーケンス

4. 形式化

4.1. 方針

特になし。

4.2. 妥当性

特になし。

4.3. 検証ツールとの相性

プロトコル仕様、攻撃者モデル、セキュリティ要件を ProVerif で記述するにあたって、特に制限はなかった。

4.4. 検証ツール適用時の性能

検証時間は 1.9 秒であった。実行環境は以下のとおり。

- ✧ Intel Core i7 L620 2.00HGz
- ✧ Windows7 上の VirtualBox 仮想マシン上の Ubuntu Linux 12.04.1 LTS
- ✧ メモリ 512MB
- ✧ ProVerif 1.86pl3

5. 備考

本文書は、**総務省「暗号・認証技術等を用いた安全な通信環境推進事業に関する実証実験の請負　成果報告書**」からの引用である。